ECE/CS 584: Embedded and cyberphysical system verification	Fall 2025
Your name, netid	*Homework 2: Dynamical systems*
	Due September 28th 11:59pm

Upload your writeup (.pdf) and code (.py) to Gradescope. In the writeup, put each answer on a separate page and label it with the correct number. Ensure your code runs on the python environment described in Problem 5. You may not get points if your code does not run. Code for this homework can be downloaded here.: https://uofi.box.com/s/ugwhbjik45exxbn10ekz9n8g12pwllel

Problem 1. (20 points). Consider any formula in difference logic (DL) $\phi(x_1, x_2, \ldots x_n)$ where ϕ is a conjunction of clauses and each clause in ϕ is of the form $x_i - x_j \leq c$, where $i, j \in \{1, \ldots, n\}$, and c is some integer constant. Develop an algorithm or a decision procedure (DP) for DL. *Recall from lecture, the idea is to construct a graph with the x_i's as vertices and then do something on that graph.*

(a) Write out the precise definition of this graph.

(b) Write the algorithm (pseudocode) for whatever your decision procedure needs to compute from this graph to output Unsat or Sat (and a model).

(c) State and prove the correctness of the (DP).

Problem 2. (20 points). Consider an automaton $\mathcal{A} = (V, \Theta, \mathcal{D})$ with Boolean variables (as in Section 7.5.3) and an unsafe set $\mathcal{U} \subseteq val(V)$ specified by a formula or predicate $\mathcal{F}_{\mathcal{U}}$ over V.

(a) How can a SAT solver be used to check bounded safety, i.e., check whether $Reach_{\mathcal{A}}(\Theta, [0, k]) \cap U = \emptyset$, for any given positive integer k? This problem is called the *bounded model checking problem* and may be relevant for your project.

(b) How does the size of the SAT problem scale with k? How does it scale with V, Θ?

Problem 3. (15 points). (a) We denote $x \in \mathbb{R}^n$ as an n-dimensional real vector, and denote $A \in \mathbb{R}^{n\times n}$ as an $n \times n$ real matrix. Then We consider a discrete-time linear automaton

$$
\begin{array}{l}
\textbf{automaton} \quad LinearSys(A : \mathbb{R}^{n\times n}, n : \mathbb{N}, K : \mathbb{N}) \\
variables: \\
\quad x : [K] \to \mathbb{R}^n \\
actions: \\
\quad step(i : [K]) \\
transitions \\
\quad step(i) \\
\quad\quad \textbf{Pre} \quad true \\
\quad\quad \textbf{Eff} \quad x_{i+1} = A \times x_i
\end{array}
$$

The set of initial states $\Theta \subset \mathbb{R}^n$ is a convex combination of a several vertices $v_0^{(1)}, \cdots, v_0^{(l)} \in \Theta$. (Convex combination means: $\forall \alpha^{(1)}, \cdots, \alpha^{(l)} \in [0,1]$ such that $\sum_{i=1}^{l} \alpha^{(i)} = 1$, if $v_0^{(1)}, \cdots, v_0^{(l)} \in \Theta$, then the state $v = \sum_{i=1}^{l} \alpha^{(i)} v_0^{(i)} \in \Theta$). Prove that the set of reachable states at step $k \in \mathbb{N}$, denoted as $\mathsf{Reach}(\Theta, [k,k])$, is a convex set. *Hint. Prove that for any $x_0 \in \Theta$, then x_k mush be contained within the convex combination of $v_k^{(1)}, \cdots, v_k^{(l)}$.*

(b) Give an example of a system for which $\mathsf{Reach}(\Theta, [k,k])$ is *not* a convex set, even though Θ is a convex set. You can give an analytical proof or a numerical example with simulations.

Problem 4. (15 points). Consider a classification neural network $y = f(x)$, where $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$. The network predicts m scores for m labels (e.g., in a traffic sign recognition neural network, the labels can be "stop sign", "speed limit sign", "yield sign", etc.), and the label with the greatest score (top-1 score) is the final predicted class of the model.

We want to verify that the prediction score of the "stop sign" is greater than that of the "speed limit sign" for all $x \in S$. If we denote the score of "stop sign" to be y_i and the score of "speed limit sign" to be y_j, we essentially want to prove that

$$y_i - y_j > 0, \quad \forall x \in S$$

where $y_i := [f(x)]_i$ and $y_j := [f(x)]_j$. In this example, we formulate this problem as a satisfiability problem, which may be checked using an SMT solver:

$$\exists x,\ x \in S \ \wedge \ (y = f(x)) \ \wedge \ (y_i - y_j \leq 0) \tag{1}$$

We verify our requirement if an SMT solver reports "unsatisfiable" for the above feasibility problem. If an SMT solver finds a satisfiable solution x_{sat}, then x_{sat} is a counterexample where the requirement $y_i - y_j > 0$ for $x_{\text{sat}} \in S$ does not hold.

In our lecture, we discussed how to use an optimization-based formulation to solve the verification problem. For example, the satisfiability problem in (1) can be solved with the following optimization problem:

$$\min_{x \in S} \ g(x) := y_i - y_j \tag{2}$$

If the objective function $g(x)$ becomes non-positive at the global optimal solution x^*, the requirement does not hold since x^* is a counterexample where $y_i - y_j \leq 0$. Otherwise, the requirement is verified.

(a) Now, you are asked to write the satisfiability problem (in a similar manner as in (1)) for verifying the following requirements:

1. The prediction score for label i is always the top-1 score for all $x \in S$
2. The prediction score for label j can never be the top-1 score for all $x \in S$

(b) Write the optimization problem that can be used to solve the above two verification requirements by redefining $g(x)$.

Problem 5. (30 points). Use the Z3 SMT solver to compute reachable set for the Dijkstra's token ring algorithm under asynchronous settings. For Z3 installation, please refer to the instructions in Homework 1, Problem 4. **Please ensure that your environment matches the test versions used on our side: `python: 3.10` and `z3: 4.8-4.15`**.

Next, download the file `Automaton.py`, `State.py`, `DijkstraSTR.py`, `DijkstraTRASyn.py` provided for this homework, put them under same folder and read them carefully. The program contains several class and functions, some of which you will need to complete. Code guidance are provided below.

`Automaton.py` defines a class named `Automaton`, which models a system with states, actions, and transitions using the Z3 SMT solver. It can check and sample initial states, see which actions are possible, and compute the next states. It also supports generating and printing execution traces, as well as building and visualizing reachability trees with matplotlib, networkx, or Graphviz. **You do not have to change it.**

`State.py` defines a class named `State`, which represents a system state as a list of (name, sort, value) tuples. It also provides methods to convert states into Z3 constraints, compare and hash states, print them for readability, and access variable values. **You do not have to change it.**

`DijkstraSTR.py` defines a class named `DijkstraSTR`, which is a specialization of class `Automaton` (defined in `Automaton.py`) that encodes Dijkstra's synchronous token ring algorithm. It models a system of N processes arranged in a ring, each holding an integer state variable modulo K. The initial predicate ensures that not all processes start with the same value, and the transition relation captures how the token is passed: if a process has the token, it updates its value (node 0 increments modulo K, while other nodes copy their predecessor's value); otherwise, it keeps its value unchanged. `has_token(self, i, x_vars)` defines the condition whether node i has the token. **You are required to implement "TODO" lines within function "has_token".**

`DijkstraTRASyn.py` defines a class named `DijkstraASYN`, which models Dijkstra's asynchronous token ring algorithm as a symbolic automaton using Z3. It represents N processes with integer state variables modulo K, where each process has its own action (mov{i}) to update its state if it holds the token. The initial predicate ensures no two adjacent processes have the same value. The transition relation enforces that only the process with the token can update its state (process 0 increments modulo K, others copy their predecessor), while all other processes remain unchanged. `transition(self, s_vars, a, s_p_vars)` encodes the transition rules of the automaton, returning a symbolic constraint that defines how the next state variables are updated based on the current state and a given action. **You are required to implement "TODO" lines within function "has_token" and function "transition".**

Submission. (a) Submit the completed file `DijkstraSTR.py` and `DijkstraTRASyn.py`. (b) In your .pdf submission, include 2 plots showing reachability tree under synchronous and asynchronous algorithm. (You should be able to generate a figure displaying the reachability tree, as shown below, by running `python3 DijkstraSTR.py`)

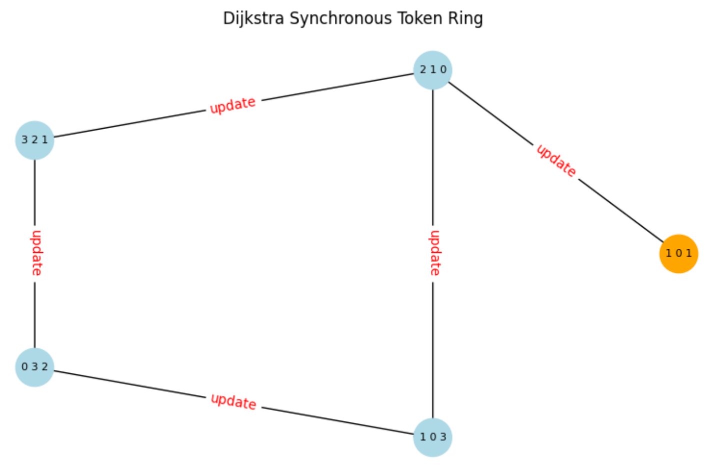

Figure 1: Sample expected output.